A FRAMEWORK FOR PERFORMING V&V WITHIN REUSE-BASED SOFTWARE ENGINEERING

Edward A. Addy
eaddy@wvu.edu
NASA/WVU Software Research Laboratory

ABSTRACT

Verification and validation (V&V) is performed during application development for many systems, especially safety-critical and mission-critical systems. The V&V process is intended to discover errors, especially errors related to critical processing, as early as possible during the development process. Early discovery is important in order to minimize the cost and other impacts of correcting these errors.

In order to provide early detection of errors, V&V is conducted in parallel with system development, often beginning with the concept phase. In reuse-based software engineering, however, decisions on the requirements, design and even implementation of domain assets can be made prior to beginning development of a specific system. In this case, V&V must be performed during domain engineering in order to have an impact on system development.

This paper describes a framework for performing V&V within architecture-centric, reuse-based software engineering. This framework includes the activities of traditional application-level V&V, and extends these activities into domain engineering and into the transition between domain engineering and application engineering. The framework includes descriptions of the types of activities to be performed during each of the life-cycle phases, and provides motivation for the activities.

INTRODUCTION

Verification and Validation (V&V) methods are used to increase the level of assurance of critical software, particularly that of safety-critical and mission-critical software. Software V&V is a systems engineering discipline that evaluates the software in a systems context. The V&V methodology has been used in concert with various software development paradigms, but always in the context of developing a specific application system. However, the reuse-based software development process separates domain engineering from application engineering in order to develop generic reusable software components that are appropriate for use in multiple applications.

The earlier a problem is discovered in the development process, the less costly it is to correct the problem. To take advantage of this, V&V begins verification within system application development at the concept or high-level requirements phase. However, a reuse-based software development process has tasks that are performed earlier, and possibly much earlier, than high-level

requirements for a particular application system.

In order to bring the effectiveness of V&V to bear within a reuse-based software development process, V&V must be incorporated within the domain engineering process. Failure to incorporate V&V within domain engineering will result in higher development and maintenance costs due to losing the opportunity to discover problems in early stages of development and having to correct problems in multiple systems already in operation. Also, the same V&V activities will have to be performed for each application system having mission or safety-critical functions.

On the other hand, it is not possible for all V&V activities to be transferred into domain engineering, since verification extends to the installation and operation phases of development and validation is primarily performed using a developed system. This leads to the question of which existing (and/or new) V&V activities would be more effectively performed in domain engineering rather than in (or in addition to) application engineering.

This paper describes a framework for performing V&V within reuse-based software. The framework identifies V&V tasks that could be performed in domain engineering, V&V tasks that could be performed in the transition from domain engineering to application engineering, and the impact of these tasks on application V&V activities. The criteria and motivation for performing V&V in domain engineering are also considered.

VERIFICATION AND VALIDATION IN TRADITIONAL SYSTEM APPLICATION ENGINEERING

V&V has been performed during application system development, within the context of many different development methodologies, including waterfall, spiral, and evolutionary development. V&V is a set of activities performed in parallel with system development and designed to provide assurance that a software system meets the operational needs of the user. It ensures that the requirements for the system are correct, complete, and consistent, and that the life-cycle products correctly implement system requirements. The V&V process evaluates software in a systems context, using a structured approach to analyze and test the software against system functions and against hardware, user and other software interfaces.

The term *verification* refers to the process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase, while *validation* is the process of evaluating software at the end of the software development process to ensure compliance with software requirements [1]. Verification is intended to ensure that the product is built correctly, while validation assures that the correct product is built.

While verification and validation have separate definitions, in practice the activities are merged into the process of *V&V*. This process evaluates software in a systems context, using a structured approach to analyze and test the software against system functions and against hardware, user and other software interfaces [2]. V&V is also described as a series of technical and

management activities performed to improve the quality and reliability of that system and to assure that the delivered product satisfies the user's operational needs [3].

V&V activities are designed to be independent of but complementary to the activities of the development and test teams. Where the development team is usually focused on nominal performance and the testing is usually based on requirements and operational profiles, V&V includes analysis and tests on critical and off-nominal behavior throughout all phases of the development lifecycle. V&V activities also complement the activities of the configuration management and quality assurance groups rather than being a duplicate or replacement of these activities [4].

A set of minimal and optional V&V activities is defined in the IEEE Standard for Software Verification and Validation Plans [5]. These activities are divided into the life-cycle phases listed below. The V&V tasks within each life-cycle phase are shown in Figure 1.

- Management of V&V
- Concept Phase V&V
- Requirements Phase V&V
- Design Phase V&V
- Implementation Phase V&V
- Test Phase V&V
- Installation and Checkout Phase V&V
- Operations and Maintenance Phase V&V

V&V is performed as a part of a risk mitigation strategy for application systems having high risk. The risks can be in areas such as safety, security, mission, financial, or reputation. The scope and level of V&V can vary with each project, based on the criticality of the system and on the role of software in accomplishing critical functions of the system[6]. V&V determines the software involved in high-risk areas, and V&V activities are focused on this critical software.

JUSTIFICATION FOR PERFORMING V&V WITHIN DOMAIN ENGINEERING

Studies have shown that the cost and difficulty of correcting an error increases dramatically as the error is discovered in later life-cycle phases[6]. V&V addresses that issue in traditional system development through activities that begin in the concept or high-level requirements phase and continue throughout all life-cycle phases. The V&V activities are focused on high-risk areas, so that errors in the high-risk areas can be discovered in time to evolve a complete and cost effective solution rather than forcing a makeshift solution due to schedule constraints.

Within reuse-based software engineering, software engineering activities may be performed prior to the concept phase of a particular application system. In order to extend the benefit of early error detection to reuse-based software engineering, V&V must be incorporated within the domain engineering process. Performing V&V at the domain level may also reduce the level of effort required to perform V&V in the individual application systems.

Although software is the target of V&V activities, V&V recognizes that software does not execute in isolation, but is an integral part of a system[7]. In order to

provide assurance that critical functions will be performed correctly, software must be evaluated within the context in which the software will execute. In reuse-based software engineering, the context for V&V must be provided by the domain model and domain architecture.

PHASE	TASKS
Management	Software Verification and Validation Plan Generation Baseline Change Assessment Management Review Review Support
Concept	Concept Documentation Review
Requirements	Software Requirements Traceability Analysis Software Requirements Evaluation Software Requirements Interface Analysis System Test Plan Generation Acceptance Test Plan Generation
Design	Design Traceability Analysis Design Evaluation Design Interface Analysis Component Test Plan Generation Integration Test Plan Generation Test Design Generation • component testing • integration testing • system testing • acceptance testing
Implementation	Source Code Traceability Analysis Source Code Evaluation Source Code Interface Analysis Source Code Documentation Evaluation Test Case Generation • component testing • integration testing • system testing • acceptance testing Test Procedure Generation • component testing • integration testing • system testing Component Test Execution
Test	Test Procedure Generation • acceptance testing Integration Test Execution System Test Execution Acceptance Test Execution
Installation and Checkout	Installation Configuration Audit V&V Final Report Generation
Operations and Maintenance	Software V&V Plan Revision Anomaly Evaluation Proposed Change Assessment Phase Task Iteration

Figure 1: V&V Tasks for Life-Cycle Phases in Application Engineering

FRAMEWORK FOR PERFORMING V&V WITHIN REUSE-BASED SOFTWARE ENGINEERING

One model for reuse-based software engineering is the Two Life-Cycle Model shown in Figure 2, developed by the U.S. Department of Defense Software for Adaptable, Reliable Systems (STARS) program. This model assumes a domain-specific, architecture-centered approach to software reuse. The domain model describes the problem space of the domain, and expresses requirements. The domain architecture describes the solution space of the domain, while the domain components are intended to be used within application systems to meet the functions described in the domain architecture.

Addy developed a draft framework for performing V&V within reuse-based software engineering engineering by adding V&V activities to the STARS Two Life-Cycle Model. The application-level IV&V tasks described in IEEE STD 1012 served as a starting point. Similar tasks that seemed appropriate were added to link life-cycle phases in the domain level, and transition tasks were added to link application phases with domain phases. This draft framework was refined by a working group at Reuse '96 [8], and the resultant framework is shown in Figure 3. The specific tasks of each phase at the domain and transition levels are listed in Figure 4.

Domain-level V&V tasks are performed to ensure that domain products fulfill the requirements established during earlier phases of domain engineering. Transition-level tasks provide assurance that an application artifact correctly implements the corresponding domain artifact. Traditional application-level V&V tasks ensure the application products fulfill the requirements established during previous application life-cycle phases.

Performing V&V tasks at the domain and transition levels will not automatically eliminate any V&V tasks at the application level. However, it might be possible to reduce the level of effort for some application-level tasks. The reduction in effort could occur in a case where the application artifact is used in an unmodifed form from the domain component, or where the application artifact is an instantiation of the domain component through parameter resolution or through generation.

Domain maintenance and evolution are handled in a manner similar to that described in the operations and maintenance phase of application-level V&V. Changes proposed to domain artifacts are assessed by V&V to determine the impact of the proposed correction or enhancement. If the assessment determines that the change will impact a critical area or function within the domain, appropriate V&V activities are repeated to assure the correct implementation of the change.

Domain-Level Tasks

The domain-level tasks are analogous to the application-level tasks, in that the products of each phase are evaluated against the requirements specified in the previous stage and against the original user requirements. The domain-level tasks can be divided into the three phases of domain analysis, domain design, and domain implementation, which correspond to the application phases of requirements, design, and implementation.

During domain analysis V&V, the V&V team should ensure that the domain model is

an appropriate representation of the user requirements. (The singular term "model" is

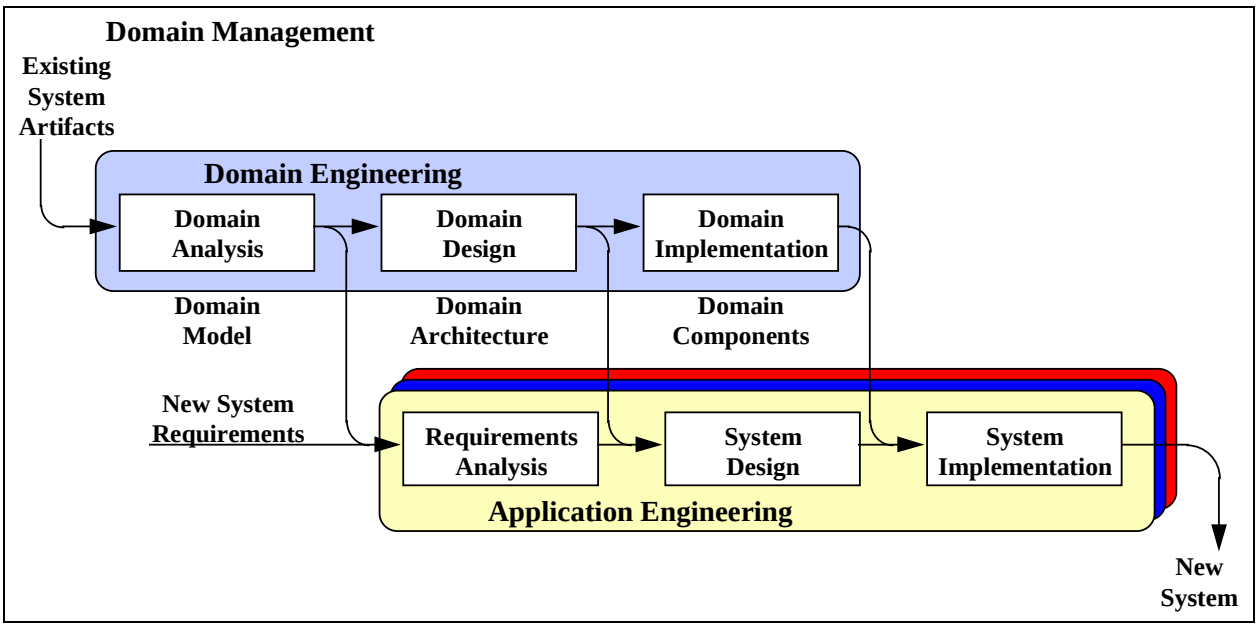

Figure 2: STARS Two Life-Cycle Model

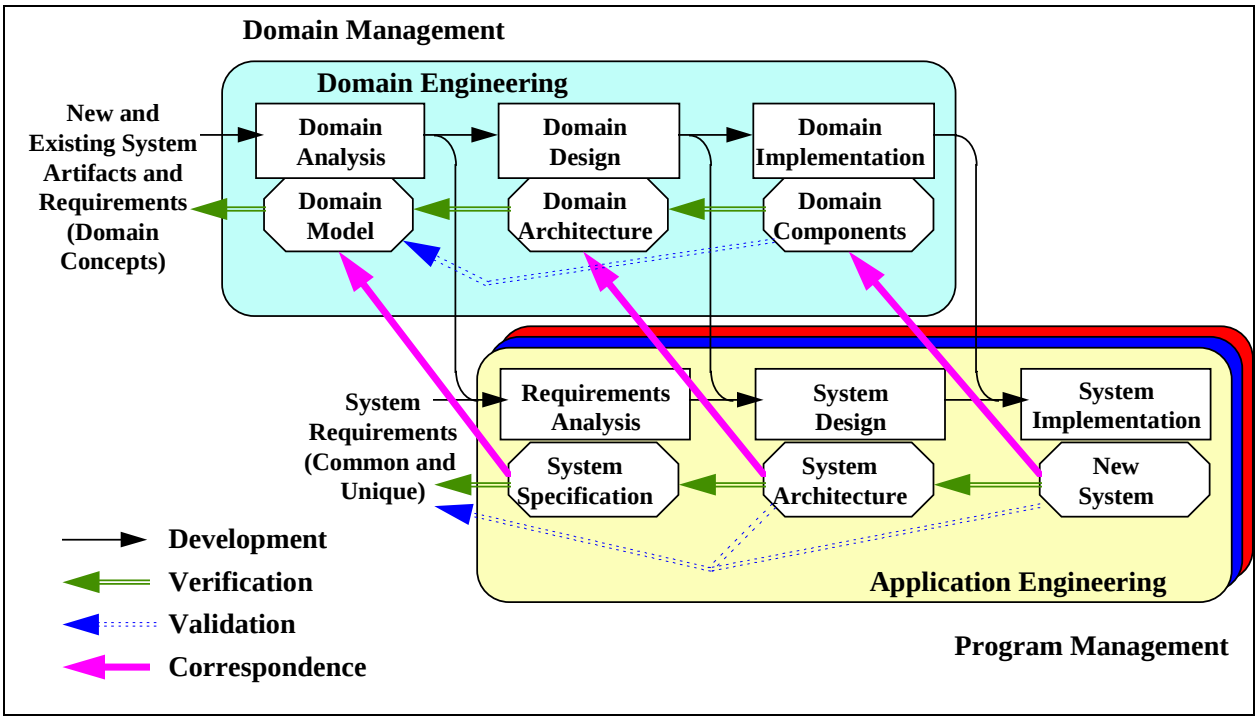

Figure 3: Framework for V&V within Reuse-Based Software Engineering

LEVEL	PHASE	TASKS
Domain Engineering	Domain Analysis	Validate Domain Model Model Evaluation Requirements Traceability Analysis (especially forward traceability for completeness)
	Domain Design	Verify Domain Architecture Design Traceability Analysis Design Evaluation Design Interface Analysis Component Test Plan Generation Component Test Design Generation
	Domain Implementation	Verify and Validate Domain Components Component Traceability Analysis Component Evaluation Component Interface Analysis Component Documentation Evaluation Component Test Case Generation Component Test Procedure Generation Component Test Execution
Transition	Requirements	Correspondence Analysis between System Specification and Domain Model
	Design	Correspondence Analysis between System Architecture and Domain Architecture
	Implementation	Correspondence Analysis between System Implementation and Domain Components

Figure 4: V&V Tasks for Life-Cycle Phases at the Domain and Transition Levels

not intended to imply that only one model will be constructed; this term is used to mean the one or more models that express the domain requirements.) Note that ensuring that user requirements are satisfied implies that the requirements in the domain must be explicitly stated. Criticality analysis is performed to ensure that high risk requirements are appropriately addressed, either mission-critical requirements or those related to properties such as safety and security. The criticality analysis should also determine critical functions that will be performed by software. The domain model is evaluated to ensure that the requirements are consistent, complete, and realistic, especially in the high risk areas. The model is evaluated to determine responses to error and fault conditions and to boundary and out-of-bounds conditions. As the domain engineering progresses into later phases, the requirements are traced forward. This will allow evaluation of the impact of changes to the domain artifacts.

Domain design V&V tasks focus on ensuring that the domain architecture satisfies the requirements expressed in the

domain model. Each requirement in the domain model should trace to one or more items in the domain architecture (forward traceability), and each item in the domain architecture should trace back to one or more requirements in the domain model (reverse traceability). The domain architecture is evaluated to ensure that it is consistent, complete, and realistic. Interfaces between components are evaluated to ensure that the architecture supports the necessary communication between components in the architecture, users, and external systems. Planning and design of component testing are performed during this phase. The component testing should include error and fault scenarios, functional testing of critical activities, and response to boundary and out-of-bounds conditions.

Domain Implementation V&V tasks ensure that the domain components satisfy the requirements of the domain architecture and will satisfy the original user requirements. The components should have a forward and reverse tracing with the domain architecture. Components that are involved with performing critical actions should receive careful consideration. The interface implementation, both within components of the architecture and with systems outside the architecture, is evaluated to ensure that it meets the requirements of the domain architecture. Component test cases and test procedures are generated, and component testing is performed.

Integration test activities are explicitly omitted from the domain-level tasking, since integration testing is oriented toward application-specific testing. Some form of integration testing might be appropriate within domain-level V&V in the case where the architecture calls for specific domain components to be integrated in multiple systems. This limited form of integration testing could be done along with the component testing activities.

Correspondence Tasks

Correspondence analysis is a term not found in IEEE STD 1012. The term is used within this paper to describe the activities that are performed to provide assurance that an application artifact corresponds to a domain artifact; i.e., the application artifact is a correct implementation of the domain artifact. Four activities are to be performed during correspondence analysis:

- Map the application artifact to the corresponding domain artifact.
- Ensure that the application artifact has not been modified from the domain artifact without proper documentation.
- Ensure that the application artifact is a correct instantiation of the domain artifact.
- Obtain information on testing and analysis on a domain artifact to aid in V&V planning for the application artifact.

Correspondence analysis is performed between the corresponding phases of the domain engineering and application engineering life-cycles. The system specification for any system within the domain should correspond to the domain model. The system specification could involve instantiating, parameterizing, or simply satisifying the requirements expressed in the domain model. Any system-unique requirements should be explicit, and the rationale for not addressing these system-unique requirements within the domain model should be stated.

The system architecture is analyzed to ensure that it satisfies the requirements specified in the domain architecture. Any variations should be documented along with the reason for the variation. The rationale for parameters chosen or options selected in constructing the system architecture from the domain architecture should be recorded.

The system components are analyzed to ensure correspondence to domain components. Again, variations, parameters, and options should be recorded along with their rationale. Baseline testing might be appropriate in order to compare variants of a domain component.

COMMUNICATING RESULTS

Communicating V&V work products and results is vital in to avoiding the repetition of V&V tasks and to ensuring that potential reusers can properly assess the status of reusable components. V&V work products and results should be associated with the component and made available to domain and application engineers. In some cases, V&V efforts might be directed at a grouping of components rather than at an individual component, and this information should also be available. Groupings might include components that are expected to occur together in several applications, or might include variants of one domain artifact.

The information on similar components within the domain should be consistent in content and format, in order to allow the information to be easily used by both domain engineers and application engineers. The information that should be communicated include the following:

- V&V Planning Decisions and Rationale
- V&V Analysis Activities
- V&V Test Cases and Procedures
- V&V Results and Findings

FUTURE WORK

Much work needs to be done to continue development of the framework for performing V&V within reuse-based software engineering. This work includes determining criteria for identifying domains where V&V is appropriate; specifying prerequisites, inputs and outputs for the domain-level and transition-level V&V tasks; and developing methods and tools to perform the domain engineering V&V tasks. Refinement of the framework will occur when experiments are conducted in applying V&V within critical domains.

CONCLUSION

The concept of V&V seems to be appropriate for reuse-based software engineering. Just as with V&V in application development, V&V should be performed as part of a risk mitigation strategy. The principle conclusions on performing V&V within reuse-based software engineering are listed below.

1. There are motivating reasons to perform V&V during domain engineering.

V&V activities might be appropriately performed during domain engineering. The primary motivation for V&V within domain engineering is to find and correct errors in the domain artifact in order to prevent the errors from being propagated to the application systems. This motivation is especially strong where the application systems perform critical functions. Even if

there are no critical functions performed by the systems within the domain, V&V might be appropriate for a component that has the potential to be used in a large number of application systems. The motivation contained within the original premise considered by the working group was that of reducing redundant V&V activities within multiple critical applications. This motivation seemed to have some merit, but appeared to be weaker than the other two reasons because of conditions described in the second finding. The reasons for performing V&V during domain engineering are listed below:

- To reduce operational risk by providing assurance that domain artifacts are correct and consistent with user needs
- To reduce the risk of a fault in a component used in many systems
- To reduce redundant V&V efforts in separate applications

2. V&V within Domain Engineering is appropriate for some domains.

V&V tasks during domain engineering will be of benefit when performed in a well-defined domain that contains multiple systems with high risk. The context in which the components will be used should be well understood, to provide a proper framework for analysis and testing of the component. The ability to perform V&V will increase as the application artifacts more closely match the domain components (e.g., unmodified reuse, application artifacts created through parameterization). The V&V effort should be tailored to address the critical areas within the domain, with the level of effort being greatest in the areas of highest criticality.

3. V&V is not appropriate in reuse outside of architecture-centered domain engineering.

Without the context of the domain, it is impossible to perform V&V activities on a component. This is consistent with the concept that V&V should consider software in relation to the system in which the software is executing. It is not possible to determine criticality or to consider the impact of fault or error conditions in isolation of context, and it is the domain architecture that provides the context for the systems in the domain.

Since general purpose reuse libraries do not typically retain the context for which the component can be reused, V&V would not generally be an appropriate activity for these libraries. This should not be understood as an argument against ensuring that domain artifacts are of a high quality and perform as described. V&V is performed within application development as a complement and not a replacement of QA and testing. QA and testing are always appropriate reuse activities, even when V&V is not possible.

REFERENCES

1. IEEE STD 729, IEEE Standard Glossary of Software Engineering, IEEE Computer Society, 1983.

2. Wallace, Dolores R. and Fujii, Roger U., Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards, NIST Special Publication 500-165, National Institute of Standards and Technology, 1989.

3. Lewis, Robert O., Independent Verification and Validation, A Life Cycle Engineering Process for Quality Software, John Wiley & Sons, 1992.

4. Wallace, Dolores R. and Fujii, Roger U., “Software Verification and Validation: An Overview”, IEEE Software, May 1989.

5. IEEE STD 1012, IEEE Standard for Software Verification and Validation Plans, IEEE Computer Society, 1986.

6. Makowsky, Lawrence C., A Guide to Independent Verification and Validation of Computer Software, Defense Technical Information Center, USA-BRDEC-TR//2516, June 1992

7. Duke, Eugene, L., “V&V of Flight and Mission-Critical Software”, IEEE Software, May 1989.

8. Addy, Edward A., “V&V Within Reuse-Based Software Engineering”, Proceedings for the Fifth Annual Workshop on Software Reuse Education and Training, Reuse ‘96, http://www.asset.com/WSRD/conferences/proceedings/results/addy/addy.html.